SALADYN

Saladyn MultiBody Toolbox

ANR Project Saladyn

ANR-08-COSI-014
Liverable 2b

Vincent Acary, Olivier Bonnefon

Date : September, 01 2010

Table of Contents

1 SALADYN Multibody toolbox task 2b - Project Overview....1
1.1 Release Information....1
1.2 Mission and Scope....1
1.2.1 What is the purpose and scope of this document ?....1
1.2.2 What is the scope of this project?....1
1.2.3 Goal of this document....1
1.3 Main features....1
1.4 Overview of existing open-source multibody toolbox....1
1.4.1 About MBDyn....1
1.5 Overview of existing collision dectection libraries....2

2 Software requirements and design. Tasks definition....3
2.1 Newton-Euler relations and dynamical systems....3
2.2 Projection on the constraints(Task Projection1)....3
2.3 Numerical solver....4
2.3.1 The bilateral constraints(Task Numerical1)....4
2.3.2 Adapt the MLCP solver(Task Numerical2)....4
2.4 CAD analysis using OpenCascade(OCC)....4
2.4.1 Initialisation of the data:....4
2.4.2 Geometrical analysis(Task OCC3)....4
2.4.3 update the OCC model(Task OCC5)....5
2.4.4 OpenCasCade(OCC) Application User Interface....5
2.4.5 Data flow....5
2.4.6 Visualisation (Task OCC4)....5
2.5 Global architecture....5
2.5.1 The input module (Task INPUT1)....6

3 Developpement plan....7

4 Appendices....9

1 SALADYN Multibody toolbox task 2b - Project Overview

1.1 Release Information

Project:	SALADYN
Internal Release Number:	1.0
Last update:	September 01, 2009

1.2 Mission and Scope

1.2.1 What is the purpose and scope of this document ?

This document describes the functionalities of the SALADYN Multibody toolbox. The major constraints of development and exploitation are delineated. The document plans the software design and development. It concerns the users and the software framework builders.

1.2.2 What is the scope of this project?

This project is a part of the SALADYN ANR project. ANR-08-COSI-014 .

1.2.3 Goal of this document

The goal of the project is to answer to the task 2b described in the project proposal document SALADYN_B

> Task 2b: Design of the multibody toolbox.

1.3 Main features

List of the main features of this module.

> Multibody mechanical systems with perfect joints.
> Unilateral constraints, impacts and Coulomb's friction.
> Collision detection.
> CAD files import and analysis.
> Salome integration.
> SALADYN toolbox integration. Model adaptability and interoperability.

1.4 Overview of existing open-source multibody toolbox

1.4.1 About MBDyn

We were interested to couple Siconos to MbDyn, because MbDyn already provides many implemented bilateral constraints (joints). Neverthelees, this coupling task would need an important effort that we are not able to do, because of our own ressources. Moreover our application needs only simple bilateral constraints (Knee, Pivot and Prismatic), so we chose to do it by ourself to obtain a self-contained toolbox in a first step.

1.5 Overview of existing collision dectection libraries

For standard multibody applications, the contact surfaces are known in advance. We only need to cimpute gap functions between two bodies and gradients of these gap functions. These functionnalities are based on the OCC methods. For the collision detection with other type if mechanical systems (FEM and granular materials), the collision detector will appear as a module of the Saladyn platform. It is planned to adapt the LMGC algorithm (Task COLL1).

2 Software requirements and design. Tasks definition

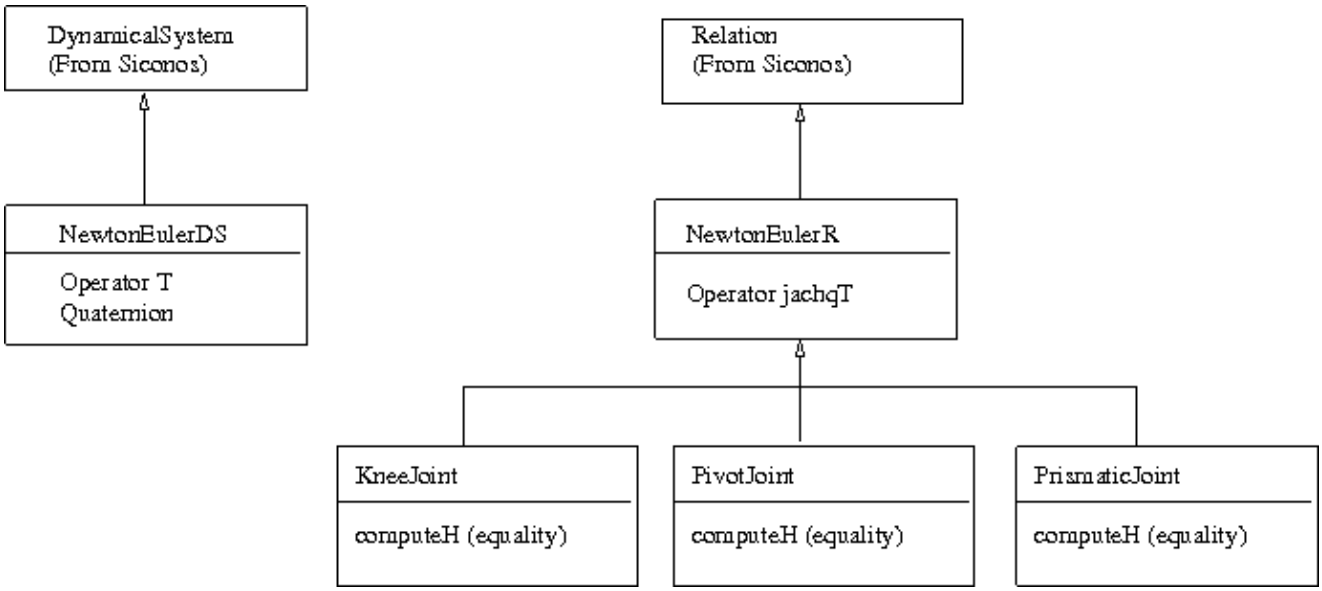
2.1 Newton-Euler relations and dynamical systems.

The multibody toolbox is based on Siconos. The holonomic constaints are defined using the standard Siconos relations. The unilateral constraints and the Coulomb friction is represented using specific nonsmooth laws. For more details, see the user documentation of Siconos.

Work to do in terms developpement: create basic joints in Siconos does not needs to much work, so it was chosen, in a first step, to develop a basic MultiBody toolbox to validate the time integration and the numerical solver.

Define a Newton-Euler representation of body in Siconos. (Task NE1)
Define relation to implement some basic joints. (Task NE2-3-4)

The following diagram shows the class hierarchy. The NewtonEulerDS adds the quaternion and the operator T to the dynamical system master class. The operator is defined by the parameterization of the finite rotations in 3D motion (see livrable L1a). The NewtonEulerR deals with the operator T for the relation. Each type of joints is inherited from the NewtonEulerR class and implements the bilateral constraints and its gradients. This equalities could be generated using a computer algebra system such as Maple or Mathematica.



2.2 Projection on the constraints(Task Projection1)

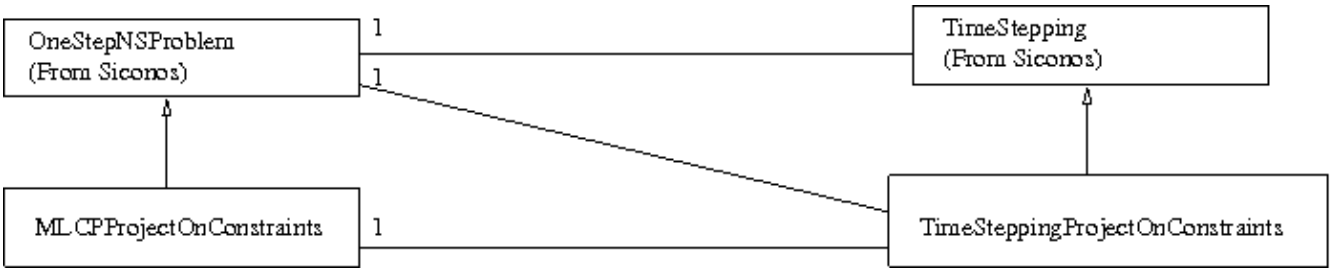
Because of the velocity formulation and its associated numerical drift, the bilateral and unilateral constraints are slight violated in discrete time. A correction is applied by adding a position formulation fo the constraints. The MLCPProjectOnConstraints class contains the formulation:

Y=h(q)
q_1 = q_0 + jachq' lambda
h(q_1)=h(q_0) + jachq (q_1 - q_0)

The obtained MLCP is:

[H(q_0)+ (jachq*jachq') lambda] \in {R,R+}*

[lambda] \in {R,R+}



The class TimeSteppingProjectOnConstraints performs this formulation during the time integration.

2.3 Numerical solver.

2.3.1 The bilateral constraints(Task Numerical1)

The current version of numerical solver only deals with friction contacts and unilateral constraints. The bilateral constraints must be added. Mainly it consists in adding the so-called GenericalMechanicalProblem including also equalities. A dedicated Gauss-Seidel algorithm will be developped is Siconos/Numerics to solve this new problem.

2.3.2 Adapt the MLCP solver(Task Numerical2)

It consists in adapting the current algorithm to manage the case where the equalities are mixed with the inequalities. The current version of Siconos/Numerics assumes that they are sorted and not mixed.

2.4 CAD analysis using OpenCascade(OCC)

This part lists the functionalities of OpenCascade that we need. We do not focus on the collision manager, but on getting the geometrical informations like, frames, axis, inertial matrix and local frame at contact.

2.4.1 Initialisation of the data:

We need to get the OCC Object (TopoDS_shape) that represents the node or the dynamical system simulated (Task OCC2). Then, we get some geometrical informations like the position, the mass, the inertial matrix and many other information to build our system.

2.4.2 Geometrical analysis(Task OCC3)

During the simulation, the SALADYN platform makes some request to OCC to get geometrical informations. From the couple of nearest points, a numerical analysis must be done to built the jacqH operator (Task OCC8).

2.4.3 update the OCC model(Task OCC5)

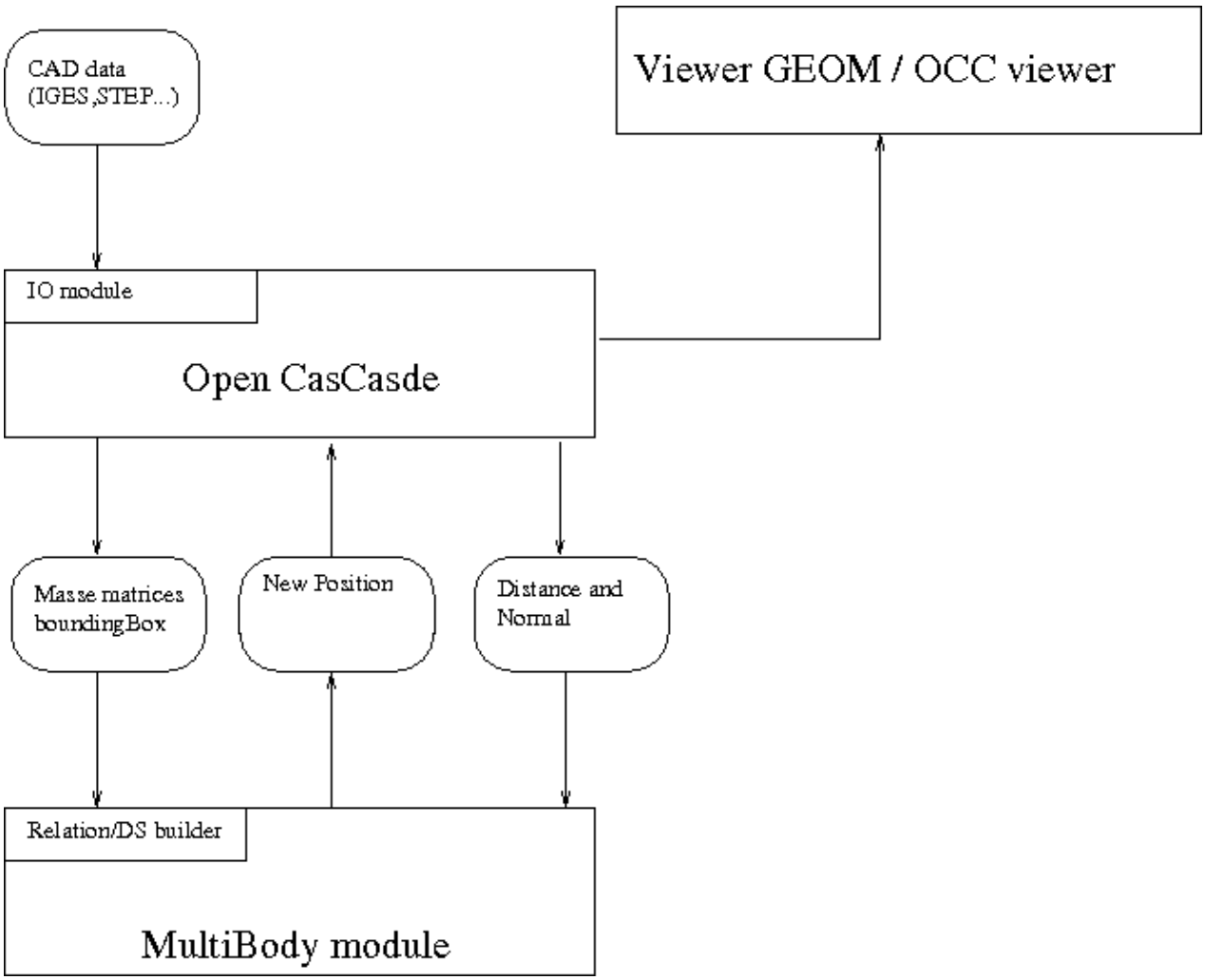
After each time-step of simulations, the positions in the OCC model must be updated. This is the goal of this tack to do it efficiently.

2.4.4 OpenCasCade(OCC) Application User Interface

The document Application user interface of OCC (see also in Appendix) contains the API waited from OCC. Several tests hav shown that OCC's algorithms to compute the nearest points between two surfaces is not efficient. It is planned to implement a new one based on the quasi-Newton methods. (Task OCC6-7)

2.4.5 Data flow

The following diagram shows the data exchanged between OpenCasCade and the multibody toolbox.

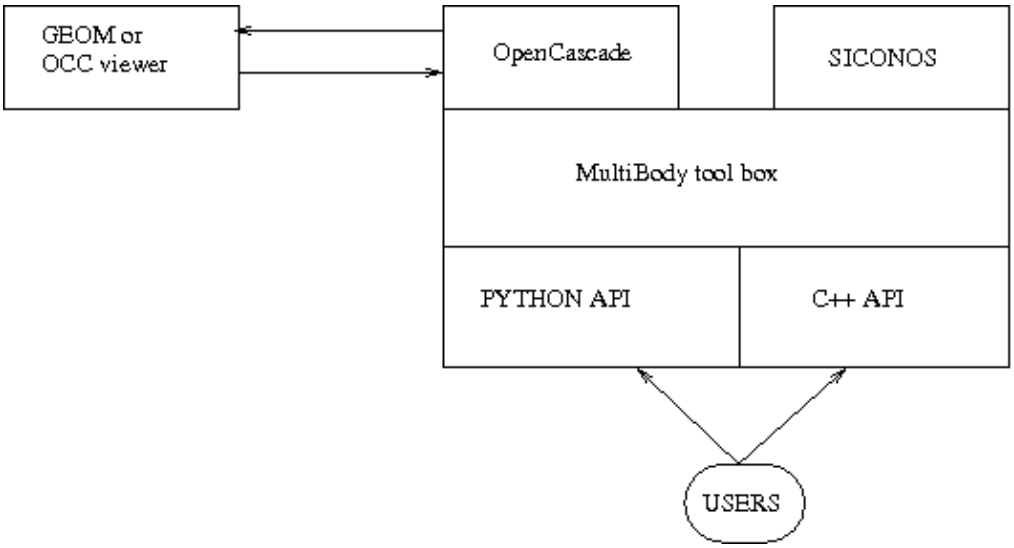


2.4.6 Visualisation (Task OCC4)

Visualisation is ensured either by Geom or an OCC viewer.

2.5 Global architecture

This section shows the relation between the module previously described. The integration to the SALADYN toolbox and then to the Salome platform will be done in Python.



2.5.1 The input module (Task INPUT1)

A task consists in writing an input module building the multi-body system from a XML file. This input file will contain the following fields:

Rigid bodies of the system.
- Id
- Name
- CAD model
- Mass and matrices
- Initial positions
- Forces and momentum plugin

Joints
- Id
- Name
- Type (Knee, prismatic ...)
- Position
- Id1 of Rigid boby
- Id2 of Rigid boby

Unilateral constraints and firction.
- Id
- Name
- Type (Friction or not)
- en
- et
- mu
- Id1 of Rigid boby
- Id2 of Rigid boby
- CAD1 model
- CAD1 model

Moreover the corresponding API will be available in C++ and in Python (Task INPUT2).

3 Developpement plan

The following table displays the time of developpement needed for each task.

Tasks Id	Tasks Summury	Estim. Days	Percent Comp.	Remark
OCC1	Training	5	100	Done with OCC staff
OCC2	IO	2	100	
OCC3	get Dist. and Normal	2	100	OCC algo is wrong
OCC4	Graphical model	2	100	Not stable
OCC5	Move an Object	3	80	
OCC6	Evaluation	3	100	
OCC7	improve dist algo	7	0	
OCC8	Build operator jacqh	5	20	
Numerical1	add a GenericMechanicalProblem	8	70	
Numerical2	Adapt MLCP solver (mixed problem)	4	100	
Projection1	projection on constraints	7	80	
NE1	add NewtonEuler Classes	5	80	
NE2	KneeJoint	2	100	
NE3	PivotJoint	2	100	
NE4	PrismaticJoint	2	100	
INPUT1	Input module	10	0	
INPUT2	API	5	0	
COLL1	Collision	15	0	

All this developpement can not be done without meeting and cordination with the other partners. For example, in view of the use case UC-62, many meeting has been organised. At present time, the multi-body toolbox is able to simulate the circuit-breaker from an initial position allowing for external forces. But some improvements must be done, mainly around the geometrical aspect (Task OCC7) and also in view of a better integration in the SALADYN platform.

4 Appendices

1. Application Programming Interface of the geometrical module of SALADYN

Application Programming Interface of the geometrical module

O. Bonnefon

September 13, 2010

This document is an initial description of the geometrical module used in the SALADYN project. The goal is to define the API of this module and to define the functionnalities waiting from OCC.

1 Project description

This section has to describe the global idea of the project. The name of the project is SALADYN. Partner are EDF, Schneider-Electric, LMGC, INRIA,LaMSID. Our goal is to simulate mechanical devices. We focus here about the geometrical aspects, two sub-projects are distinguished in the followinf subsections.

1.1 Using GEOM

In this case, the loading and the viewing is ensured by GEOM (included in SALOME). From GEOM (or from OCC?), we need:

- To get the OCC objects (surfaces, frames,...).
- To compute distances and points of contacts.
- To set and get positions of objects.
- To send a signal to GEOM to redraw the scene.
- To add simple graphical objects (lines and points) to see the interactions between the objects.

1.2 Using a simple viewer based on OCC

We also want to develop a sample viewer based on OCC (and Qt for example) to perform the mechanical simulation. Mainly it consists in using only OCC to load and explore the topology, to compute distance between objects and to update the scene during the simulation.

2 Application Programming Interface

This section has to describe the API used by the other modules (SICONOS-LMGC-SALADYN). Some of these functions will be implemented using OCC.

2.1 Detection collision

Get the bounding box of an object.

1. void getBoundingBox(in GeometricalModel, out Pinf, out Psup)

2.2 Punctual contact. Nonconforming contact

In the case of convex shapes, it consists in computing the couple of points where the minimal distance is reached, see Fig 1.

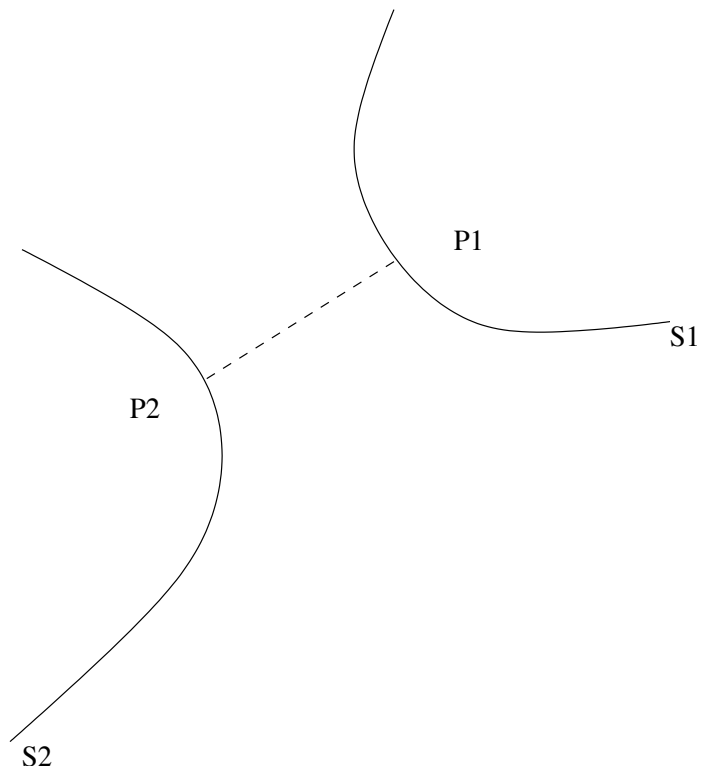


Figure 1: Simple case. {fig:simpleCase}

In a more general case, for each part of two objects nearer than a given level, we need a couple of points, see Fig 2.

In the case of a unique point of contact, the API could be:

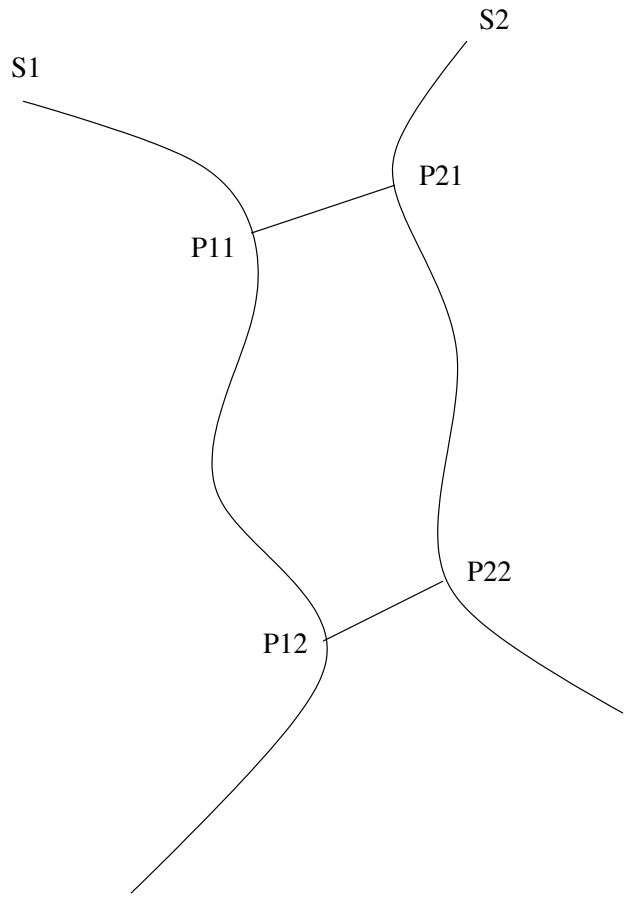


Figure 2: Multiple interaction. {fig:MultiCase}

1. double getNearestPoint(in GeometricalModel G1,in GeometricalModel G1, out P1, out P2), returning the distance between points.

About the case of multi-contact the following API is proposed :

1. void computeContacts(in GeometricalModel G1,in GeometricalModel G2,in double dist)
2. int getContactCouple(out Point P1, out Point P2) return 1 while the list of punctual contact is not empty, else return 0.

Algorithm 1 Example 1

```
Require: G1 G2, Geometrical models.
  computeContacts(G1,G2,0.1)
  while getContactCouple(P1,P2) do
    Do what you want using P1, P2.
  end while
```

{algo:example1}

2.3 Non punctual interaction. Conforming contact

In the case of a 2D interactions between objects, we distinguish two strategies: The first consists in computing the intersection (or an approximation), the second one is based on the computation of a common grid.

2.3.1 Compute the intersection

It consists in computing the geometrical intersection between two shapes. For example, the list of the closed-wire resulting of the intersection of two objects.

1. void computeWireContact(in GeometricalModel G1,in GeometricalModel G2)
2. int getWire(out Wire W), return 1 while the resulting list is not empty, else return 0. W is a wire (data structure that must be defined) resulting of the intersection computation.

Algorithm 2 Example 2

```
Require: G1,G2 Geometrical model.
  computeWireContact(G1,G2)
  while getWire(W) do
    Do what you want using W.
  end while
```

{algo:example2}

2.3.2 compute a grid

It consists in computing a 2D grid separating two objects. The idea consists in representing the interactive zones between objects like a set of points.

1. void computeGrid(In GeometricalModel G1,In GeometricalModel G2, In double dist, In double gridTol ,....) where
   (a) dist represents the thickness of the interactive zone.
   (b) gridTol is the accuracy of the computed grid.
   (c) ... must be defined, depending of the grid builder algorithm.
2. void getGridCouple(out P1, out p2, out id) return 1 while the resulting list is not empty, else return 0. where

(a) P1 is a 3d point

(b) P2 is a 3d point.

(c) id identify a couple.

Algorithm 3 Example 3

```
Require: G1 G2 Geometrical model.
  computeGridContact(G1,G2,0.1,...)
  while getCoupleGrid(P1,P2,id) do
    Do what you want.
  end while
```

{algo:example3}

Question for LMGC and SICONOS: Is the grid plane ?

3 Functions waited from OpenCascade

This section lists the functionalities waited from OCC. These functions will be used to implement the geometrical module of SALADYN.

3.1 Basic needs

Following, the list of the essential functionalities:

- getBoudingBox(in TopoDS O,in P1,out P2): It computes the bounding box of an object.
- getNearestPoint(in TopoDS O1, in TopoDS O2, P1, P2): It computes the couple of points where the minimal distance is got.
- Set and get Position(TopoDS O1,in out Position): To set and get the position (and orientation) of an object.
- computeProjection(In TopoDS O1, in P1, out P2): It consists in projecting the 3D point P1 on the surfaces 01.
- How explore the topology after importing a step file?
- Build and delete basics geometrical objects.
- computeIntersection(TopoDS 01, TopoDS 02, out ListOfTopoDS_Wire): compute the list of wires resulting to the intersection of O1 and O2.
- In the case of qt application, redraw the scene.
- Get a normal at a point located on a surface.

3.2 Other functionalities

Following, the list of secondary functionalities.

- Offset a surface: Minkowski sum. (it could be a pre-computation done by GEOM)
- Get a couple of point for all interactive zones (see Fig. 2).

3.3 Questions

- Are there some 2D parameter represented a surface ?
- Is it possible to use a name or an identifier to exchange surfaces, wires or points between software?
- How is located a point on a surface?